Query Language and Engine and Identifiers
Position Paper
DRAFT 1.0

Scott Oster and Tahsin Kurc
Ohio State University
01/04/2005

Abstract

This paper will briefly describe the impact the common query language and engine will have on the identifier requirements. We consider two main types of identifiers: resolvable and physical (non resolvable). For the purposes on this paper we will consider a resolvable identifier to be one which must be "resolved" to locate the actual physical location of the data. A physical identifier is one which can directly be used to access data (such as a URI).

Resolvable Identifiers

If resolvable identifiers are used, there must be a service which can be used to resolve an identifier into a physical identifier. While it is expected that a query service will be able to return data given either a physical or resolvable identifier, it is not expected that a query service will be the primary service utilized for identifier resolution.

It is also not expected that identifiers will be the only mechanism to perform joins between databases. Consider a user who wants to do comparisons between two patient populations and that the information collected on the two populations is stored in separate databases. The user may want to submit a query that will do a join between the databases based on geographic location (e.g., county). The user may also want to execute a query that will perform a join on lab results (e.g., find all the patients from the two populations where the blood pressure is the same). In these cases, the user does not explicitly make use of identifiers, though it is likely the query engine will make extensive use of them in carrying out the query execution plan. It can be expected that in many cases the end user will not remember the identifiers or care about specifying identifiers in a query.

If the query engine is expected to retrieve remote data from resolvable identifiers when performing joins and there are multiple possible redundant physical locations for the data, the resolution service should support the ability to specify resolution preferences. For example, the query engine will certainly prefer pulling data from local sources, and may prefer to increase or limit the number of involved data services in a join if possible.

It is currently unclear how identifiers will be used to store data in the presence of both resolvable and physical identifiers. If data services use both to store references to remote data, it may make identifier comparison difficult for query engines as it would not be able to do a simple "=" operation to compare a physical identifier with a resolvable one. It

also bears mentioning some resolving schemes may not support reverse lookup of physical identifier to resolvable identifier. Furthermore, there may be a "one to many relationship" in either direction, which significantly increases the complexity of comparing these two types of identifiers with each other.

Physical Identifiers

If non-resolvable, physical identifiers are used, data will be more tightly coupled to the service it is stored in, but query engines will have less complexity to deal with.

Both resolvable and physical identifiers share some characteristics with respect to their impact on query engines. If identifiers are used as foreign keys in one data service, to point to data in another data service, it is expected that the query language should support retrieval and introspection of the remote data. From a performance perspective, we need to have support for caching resolved identifiers and compact, machine-readable representations of identifiers. Caching is important as if the query engine has to contact the identifier resolution service for each result returned from the data source, execution of queries will take too long in many cases. Another approach could be to have a compact representation (e.g., 64-bit or 128-bit number) for each identifier and store this information in the database as part of the data element. In that case, the query engine uses the compact representation directly when performing joins and can retrieve the resolvable identifier from the identifier resolution service when needed.

If a data service contains a collection of identifiers that, for example, point to Genes, one should be able to express a query that says "Give me the Genes that have the name = `BRCA1`". In order to support this, the query engine must be aware that the data stored are identifiers. This implies that a common data type must be defined for identifiers and leveraged by data services. An alternative approach would be to not place restrictions on the data types, but introduce a `resolve(id)` operator (or a similar operator) into the query language which could be used to explicitly interpret its argument as an identifier.

Another facet of identifiers which will greatly impact query engines is the amount of information that is encoded into them directly. It has been proposed that various aspects could be encoded into the identifier itself, such as: data type, version number, concept forking and merging, and other various metadata. If such things are encoded in the identifiers, undoubtedly the query engines will be expected to utilize such information when executing queries. This seems attractive in that remote services do not need to be contacted to attain such information, but there is a balance, as too much implied information may put a heavy burden on the query engine. For example, when concepts like identifier fork and merge are supported, these semantics will probably be expected to be respected when performing joins. For example, if an identifier has been forked, and a query is executed, should the query engine be expected to search for each of the resultant children identifiers when performing an equi-join on the original identifier? It is our opinion such things should be handled explicitly in either the data or a semantic layer should support representation of such concepts, rather than implicitly in the query execution.